PATENT APPLICATION

WEB SERVER IN-KERNEL INTERFACE TO DATA TRANSPORT SYSTEM AND CACHE MANAGER

By Inventors:

Sunay Tripathi
941 Live Oak Drive
Santa Clara, CA 95051

Bruce W. Curtis
1247 Lakeside Drive 2038
Sunnyvale, CA 94086
Citizenship: U.S.

Assignee: Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto, CA 94303

Entity: Large

BEYER & WEAVER, LLP
P.O. Box 61059
Palo Alto, CA 94306
Telephone (510) 843-6200

WEB SERVER IN-KERNEL INTERFACE TO DATA TRANSPORT SYSTEM AND CACHE MANAGER

5

Cross Reference to Related Applications

10   This invention is related to U.S. Patent Application No. 09/513,320 (attorney docket no. SUN1P702/P4739), filed on February 25, 2000, naming Bruce W. Curtis as inventor, and entitled "USER LEVEL WEB SERVER IN-KERNEL NETWORK I/O ACCELERATOR." That application is incorporated herein by reference in its entirety and for all purposes.

15

          This invention is related to U.S. Patent Application No. 09/513,328 (attorney docket no. SUN1P701/P4732), filed on February 25, 2000, naming Bruce W. Curtis as inventor, and entitled "USER LEVEL WEB SERVER CACHE CONTROL OF IN-KERNEL HTTP
20 CACHE." That application is incorporated herein by reference in its entirety and for all purposes.

25   ## BACKGROUND OF THE INVENTION

1.   FIELD OF THE INVENTION

          The present invention relates generally to computer software. More

particularly, the present invention relates to methods and apparatus for

providing an in-kernel interface to a web server.

30

2.   DESCRIPTION OF RELATED ART

          FIG. 1 is a block diagram illustrating a conventional web server 104.

Through the use of a web browser and the web server 104, a user may access a

web page stored on the web server 104 through the Internet. In this manner,

35 multiple Hypertext Transfer Protocol (HTTP) clients (e.g., web browsers) 100,

102 may access files via the single web server 104. Typically, a browser user

enters HTTP file requests by either "opening" a Web file (e.g., typing in a Uniform Resource Locator or URL) or clicking on a hypertext link. The browser builds a HTTP request and sends it to the Internet Protocol (IP) address indicated by the URL. When the web browser 100 or 102 sends a HTTP request to the web server 104 identified by the IP address, the web server 104 receives the request and, after any necessary processing, the requested file (i.e., HTTP response data) is returned.

Within the web server 104, HTTP requests that are received are processed by a HTTP daemon 105. The HTTP daemon 105 is a program that runs continuously on the web server 104 and exists for the purpose of handling HTTP requests. The HTTP daemon 105 forwards the received HTTP requests to other programs or processes as appropriate. Thus, each web server has a HTTP daemon 105 that continually waits for requests to come in from Web clients and their users. Once a file (i.e., HTTP response data) is obtained (e.g., from an associated web server cache memory 106), the data is transmitted by the daemon 105 to the client 100 or 102 that requested the data. In addition, the web server cache 106 is often used to store HTTP response data. As an alternative to cache memory, the HTTP daemon 105 may have other storage media associated with it. Such media, for example, can include a hard drive.

HTTP requests are typically initially handled by a kernel 107 that is responsible for forwarding the requests from the client 100 or 102 to the HTTP daemon 105. The kernel 107 is the essential central part of a computer operating system, the core that provides basic services for all other parts of the operating system. Typically, a kernel includes an interrupt handler that handles all requests or completed I/O operations that compete for the kernel's services, a scheduler that determines which programs share the kernel's

processing time in what order, and a supervisor that actually gives use of the computer to each process when it is scheduled. The kernel 107 may also include a manager of the operating system's memory address spaces, sharing these among all components and other users of the kernel's services. A

5   kernel's services are requested by other parts of the operating system or by applications through a specified set of program interfaces sometimes known as system calls. The kernel 107 also provides services such as buffer management, message routing, and standardized interfaces to protocols which enable data to be routed between clients 100, 102 and a server 104.

10   As it applies to handling server/client communications, the kernel structure consists of three layers: a socket layer 108, a protocol layer 110, and a device layer 111. The socket layer 108 supplies the interface between the HTTP daemon 105 and lower (protocol and device) layers, the protocol layer 110 contains protocol modules for communication, and the device layer 111

15   contains device drivers that control network devices. Thus, a server and client process may communicate with one another through the socket layer 108. More particularly, a socket file system 109 (SOCKFS) is associated with the socket layer 108 and is adapted for managing the socket layer 108.

Conventional Unix network input/output is provided through the use

20   of a file descriptor opened on a socket. A "socket" is a method for communication between a client program and a server program in a network. A socket is defined as "the endpoint in a connection." Sockets are created and used with a set of programming requests or "function calls" sometimes called the sockets application programming interface (API). A file descriptor

25   is typically an integer that identifies an open file within a process which is obtained as a result of opening the file. In other words, a separate socket is required for each network connection. Thus, as shown, each network

connection corresponding to a client request has an associated socket layer 112 and protocol layer 114, which may send data via a network interface card 116 via a transmission medium 118 to one or more clients 100, 102. Each socket has its own socket data structure. Since a separate file descriptor is

5   opened on a socket for each network connection, in-kernel resources are unnecessarily consumed. Moreover, there are limits to the number of file descriptors that may be opened at a particular instant in time.

STREAMS is a general, flexible programming model for Unix system communication services. STREAMS defines standard interfaces for character

10 input/output (I/O) within the kernel, and between the kernel and the rest of the UNIX system. The mechanism consists of a set of system calls, kernel resources, and kernel routines. STREAMS enables the creation of modules to provide standard data communications services. A STREAMS module is a defined set of kernel-level routines and data structures. From the application

15 level, modules can be dynamically selected and interconnected. No kernel programming, compiling, and link editing are required to create the interconnection. STREAMS provides an effective environment for kernel services and drivers requiring modularity. STREAMS parallels the layering model found in networking protocols.

20 A stream is a data path that passes data in both directions between a STREAMS driver in kernel-space and a process in user space. An application creates a stream by opening a STREAMS device. When a STREAMS device is first opened, the stream consists of only a stream head and a STREAMS driver. A STREAMS driver is a device driver that implements the STREAMS

25 interface. A STREAMS device driver exists below the stream head and any modules. It can act on an external I/O device, or it can be an internal software driver, called a pseudo-device driver. A stream-head is the end of the stream

nearest the user process. It is the interface between the stream and the user process. The STREAMS device driver transfers data between the kernel and the device. STREAMS enables the manipulation of the modules on a stream.

In order for the TCP protocol layer to communicate with the HTTP daemon, a new stream is typically created for each connection. Since a stream is associated with a single connection, the stream does not include identifying information that identifies the connection. On the contrary, since a separate stream is opened for each connection, such identifying information is stored in association with the connection (e.g., by the TCP protocol layer and by the SOCKFS). This private state which uniquely identifies the connection includes information such as a remote IP address, a remote port, a local IP address, and a local port. It is important to note that since such identifying information is not included in the stream, data for only a single connection may be sent in the stream. As a result, multiple streams must be created in order to transmit HTTP request data from a client to the HTTP daemon. Since it is difficult to pre-create such streams, this stream creation is preferably performed dynamically. However, numerous steps must be performed before data can be sent in a data stream.

FIG. 2 is an exemplary diagram illustrating data flow associated with a HTTP request. TCP protocol layer is represented by line 202, socket file system (SOCKFS) is represented by line 204, and HTTP daemon is represented by line 206. As shown, for each connection 208, the TCP protocol layer 202 sends a connection indication 210 to the socket file system 204. The SOCKFS 204 then sends a connection indication 212 to the HTTP daemon 206. The HTTP daemon 206 accepts the connection by sending a message as shown at 214 to the SOCKFS 204. The SOCKFS then sends an acknowledgement 216 to the TCP protocol layer 202. Once the

acknowledgement 216 has been received by the TCP protocol layer 202, data 218 is sent to the SOCKFS 204. The data is then transmitted as shown at 220 to the HTTP daemon 206. The HTTP daemon 206 then accepts the data as shown at 222. The SOCKFS 204 then sends an acknowledgement 224 to the

5   TCP protocol layer 202. As shown and described with reference to FIG. 2, multiple steps must typically be performed in order to transmit HTTP request data to a web server. Accordingly, it would be beneficial if the time required to send HTTP request data to a web server could be reduced while minimizing memory and processing resources. Similarly, it would be desirable if HTTP

10   response data could be sent in a compatible and efficient manner from the web server to the requesting client. In addition, it would be preferable if such a system could be implemented without requiring modifications to the HTTP daemon.

SUMMARY

The present invention enables HTTP requests to be transmitted to a web server while minimizing memory and processing resources. Similarly, HTTP response data is sent in a compatible and efficient manner from the web server to the requesting client.

In accordance with one aspect of the invention, a HTTP request is sent to a web server. A HTTP request including HTTP request data is received. A connection identifier is associated with the HTTP request. The receiving and associating steps are repeated for one or more HTTP requests. The connection identifier and the associated HTTP request data for the one or more HTTP requests are then sent in a single stream to the web server (e.g., HTTP process).

In accordance with another aspect of the invention, a HTTP response including HTTP response data received from a web server is processed. HTTP response data and an associated connection identifier are received from a HTTP process. A stream is created. The HTTP response data and the associated connection identifier are then sent in the stream.

In accordance with another aspect of the invention, a HTTP request including HTTP request data are processed by a web server. HTTP request data and an associated connection identifier are received. HTTP response data associated with the HTTP request data is obtained. The HTTP response data and the connection identifier are then sent.

Brief Description of the Drawings

The invention, together with further advantages thereof, may best be understood by reference to the following description taken in conjunction with the accompanying drawings in which:

FIG. 1 is a block diagram illustrating a conventional web server.

FIG. 2 is a process flow diagram illustrating conventional stream creation and management of HTTP request data.

FIG. 3 is a block diagram illustrating a system in which a network cache accelerator and associated file system are implemented in accordance with an embodiment of the invention.

FIG. 4 is a process flow diagram illustrating stream creation and management of HTTP request data in accordance with an embodiment of the invention.

FIG. 5 is a process flow diagram illustrating a method of configuring a web server in accordance with an embodiment of the invention.

FIG. 6 is a process flow diagram illustrating a method of processing a HTTP request in accordance with an embodiment of the invention.

FIG. 7 is a process flow diagram illustrating a method of processing a HTTP response in accordance with an embodiment of the invention.

FIG. 8 is a block diagram illustrating an exemplary object that may be used to transport HTTP request and response data in accordance with an embodiment of the invention.

FIG. 9 is a block diagram illustrating a typical, general-purpose computer system suitable for implementing the present invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

In the following description, numerous specific details are set forth in order to provide a thorough understanding of the present invention. It will be apparent, however, to one skilled in the art, that the present invention may be practiced without some or all of these specific details. In other instances, well known process steps have not been described in detail in order not to unnecessarily obscure the present invention.

In the following described embodiment, HTTP request and response data are transferred between a HTTP daemon (i.e., web server) and a network cache accelerator. Moreover, the present invention may be implemented using any appropriate mechanism (e.g., Remote Procedure Call mechanism) for communicating between an application and an in-kernel module. For instance, the present invention may be applicable in a Unix system and makes use of the Unix notion of the filesystem as a universal name space. In addition, the present invention may implement a Remote Procedure Call (RPC) mechanism which has built in support for multi-threading. The present invention may be implemented on any system which includes an application and a kernel. For instance, the invention may be applicable to a system having a kernel and an application transport protocol layer (e.g., FTP) which is data intensive.

FIG. 3 is a block diagram illustrating a system in which a network cache accelerator and associated file system are implemented in accordance with an embodiment of the invention. As shown in FIG. 3, multiple clients

100, 102 may send HTTP requests to a web server 302. Within the web server 302, a network cache accelerator (NCA) 304 includes a cache manager 305 and a data transport module 306. An in-kernel cache 307 is managed by the cache manager 305 which has an associated protocol stack 308. The data

5   transport module 306 routes HTTP requests or portions thereof (and/or other information or requests) to a HTTP daemon 310 via a NCA file system 312. More particularly, the data transport module 306 places an object (e.g., containing the HTTP request data, connection identifier and/or other requests or information) for one or more HTTP requests in a single listening stream.

10   As described above, the prior art method sent a connection indication to SOCKFS, which would forward it to the web server. The web server would then accept the connection and a new stream would then be created between the SOCKFS and the web server. Once the stream was created, the data could be sent to the web server via the SOCKFS.

15   The NCA file system 312 preferably stores the HTTP request data and associated connection identifier for the HTTP requests. The HTTP daemon 310 obtains the HTTP request data from the NCA file system 312. The connection identifier is preferably maintained as private information between the NCA and the NCAFS to enable the present invention to be used with any

20   web server. In other words, the web server need not be modified to accommodate the present invention.

Next, a method implemented by the HTTP daemon 310 is invoked and the HTTP daemon 310 returns a HTTP response (or portion thereof) and/or directives to control information that is stored in the in-kernel cache 307 or

25   control the transmission of information to a client 100 or 102. This information is sent to the NCA 304. More particularly, the HTTP daemon 210 provides an object containing the HTTP response and/or directives to the

NCA file system 312. The NCA file system 312 obtains the object, provides

the connection identifier in the object, creates a new stream, and sends this

object to the NCA 304 via the newly created stream. It is important to note

that the new stream may be created while the web server 302 is processing the

5   request. The NCA 304 may then obtain the HTTP response and/or directives

from the object so that it may determine how to manage the transmission

and/or storage of response data received from the HTTP daemon 310. In this

manner, the HTTP daemon 310 may manage information that is stored,

modified and/or purged from the in-kernel cache 307 as well as control

10   information that is transmitted to the clients 100 and 102.

FIG. 4 is a process flow diagram illustrating stream creation and

management of HTTP request data in accordance with an embodiment of the

invention. As shown, the NCA is represented by vertical line 400, the NCA

file system is represented by vertical line 402 and the HTTP daemon is

15   represented by vertical line 404. When a connection 406 (i.e., HTTP request)

is received by the web server, the NCA 400 sends a connection identifier and

HTTP data for the HTTP request as shown at 408. More particularly, the

NCA 400 sends a connection identifier and associated HTTP data for one or

more HTTP requests in a single stream. This is possible since the amount of

20   data that must be sent for a HTTP request is relatively minimal in comparison

to a HTTP response. In other words, a new stream need not be created for

each new HTTP request received. The NCA file system 402 stores the

connection identifier and HTTP request data for the HTTP requests as shown

at 410. For each HTTP request, the NCA file system 402 sends a connection

25   indication 412 to the HTTP daemon 404. The HTTP daemon sends an accept

command 414 followed by a read request as shown at 418 to the NCA file

system 402. Since the NCA file system 402 has already obtained the HTTP

request data for the HTTP request, the NCA file system responds by sending the associated HTTP request data at 420 to the HTTP daemon 404.

The HTTP daemon 404 then prepares the response 422. As shown at 416, the NCAFS establishes a stream to send the HTTP response prepared at 422. More particularly, the stream may be established in parallel with steps 418 and 422. Once the response is prepared at 422, the response is sent to the NCAFS at 424. At 426 the NCAFS 402 then sends the response and associated connection identifier over the stream established at 416.

In accordance with one embodiment, the connection identifier is sent to the HTTP daemon with the HTTP request as well as from the HTTP daemon with the HTTP response. More particularly, when the HTTP daemon 404 accepts the connection as shown at 414, a new file descriptor is returned to it. It then "reads" the HTTP request on this new file descriptor. In accordance with the present invention, a segment of private information is attached to this new file descriptor. More particularly, the private information includes a connection identifier associated with the HTTP request. The private information is intended to be shared by the NCA and the NCAFS to identify the HTTP request and therefore need not be obtained or examined by the web server. When the HTTP daemon sends the HTTP response to the NCAFS at 424, this "write" is performed on this same file descriptor on which the "read" was performed. In this manner, the NCAFS may obtain the connection identifier related to that connection from the private information such that the HTTP response may be associated with the previously sent HTTP request.

FIG. 5 is a process flow diagram illustrating a method of configuring a web server in accordance with an embodiment of the invention. As shown, the web server is booted up at block 502. A permanent listening stream is

then created at block 504. Once the permanent listening stream is created, the connection identifier and associated HTTP request data for one or more HTTP requests may be sent in the permanent listening stream to the web server or HTTP process such as a HTTP daemon. In this manner, HTTP request data

5   for multiple HTTP requests may be sent simultaneously for access by a web server. When the web server is shut down, the listening stream may then be removed.

Once the web server is set up, HTTP requests may be sent to the web server. Similarly, once the HTTP requests are processed by the web server,

10   HTTP responses may be sent to the requesting clients. Processing of HTTP requests and responses will be described in further detail below with reference to FIG. 6 and FIG. 7, respectively.

FIG. 6 is a process flow diagram illustrating a method of processing a HTTP request in accordance with an embodiment of the invention. When a

15   connection is received by the web server, a HTTP request including HTTP request data is received at block 602. A connection identifier is then associated with the HTTP request at block 604. If it is determined that there are more HTTP requests at block 606, steps 602 and 604 are repeated for one or more HTTP requests. Thus, the NCA continually sends HTTP requests as

20   and when they arrive. In other words, the NCA need not wait for more HTTP requests to be received. At block 608, the connection identifier and the associated HTTP request data for the one or more HTTP requests are sent (e.g., by the NCA) in a single stream such as the permanent listening stream described above with reference to FIG. 5. For instance, the connection .

25   identifier and the associated HTTP request data may be provided to a cache manager capable of storing the connection identifier and the associated HTTP request data and retrieving the HTTP request data when the connection

identifier is received. The connection identifier and the associated HTTP

request data for the one or more HTTP requests may be provided in an object

which is then sent to a HTTP process. Alternatively, the connection identifier

and the associated HTTP request data for each of the one or more HTTP

5   requests may then be stored temporarily (e.g., by the NCA file system) at

block 610 for retrieval by the HTTP process (i.e., HTTP daemon). The cache

manager may therefore store the HTTP request and response data so that the

same request may be handled by the NCA file system in the future. In order

to notify the HTTP process that a connection has been received and that HTTP

10   data has been stored for retrieval by the HTTP process, the connection

indication for one of the HTTP requests is sent to the HTTP process at block

612.

       When the HTTP process receives the connection indication for a

HTTP request, the HTTP process accepts the connection and sends a read

15   command at block 614. In response, the HTTP request data associated with

the connection identifier received from the HTTP process is obtained at block

616. The obtained HTTP request data and the associated connection identifier

is then sent to the HTTP process at block 618. If it is determined at block 620

that there are more HTTP requests, the process repeats at block 612.

20   Otherwise, the process ends at block 622.

       FIG. 7 is a process flow diagram illustrating a method of processing a

HTTP response in accordance with an embodiment of the invention. The web

server (i.e., HTTP process) processes the HTTP request to obtain HTTP

response data associated with the HTTP request data at block 702. The HTTP

25   process then performs a write by sending HTTP response data to the NCAFS

at 703. The NCAFS then packages the HTTP response data with the

connection identifier identifying one of the HTTP requests received by the

web server at block 704. When the HTTP response data has been obtained

from the HTTP process, a new stream is created at block 706. For instance,

the new stream may be created by the NCA file system. The HTTP response

data and the associated connection identifier are then sent in the newly created

5   stream at block 708. For instance, an object may be instantiated in which the

HTTP response data and the connection identifier are provided. The NCA

(e.g., data transport module) may then intercept and transmit the HTTP

response data to the client.

As described above, the NCA 304 and the HTTP daemon 310

10   communicate through sending an object. FIG. 8 is a block diagram

illustrating an exemplary data type that may be transported in accordance with

an embodiment of the invention. More particularly, in accordance with one

embodiment, the data transport module 306 and the HTTP daemon 310 both

transmit a HTTP request-response object. The information that may be

15   provided in the HTTP request-response object is illustrated generally in FIG.

8. A HTTP request-response object 802 is shown to identify the data stream

through an identifier ID 804 between the data transport module and the HTTP

daemon. In addition, the HTTP request-response object 802 may transmit

either a HTTP request or a HTTP response. More particularly, a HTTP_OP

20   field 806 indicates whether the object is transmitting request or response data.

In addition, the HTTP_OP field 806 may also indicate that an error message is

being transmitted. A FIRST field 808 indicates whether this is the first block

of data in a set of request or response data. A MORE field 810 indicates

whether more request or response data is to follow for the associated HTTP

25   request or response, respectively.

In accordance with one embodiment, the HTTP request-response

object 302 also transmits information that can modify the flow of data

15

between the data transport module and the HTTP daemon as well as the flow of data to the client. A PREEMPT field 812 may be set to preempt data flow from the data transport module to the HTTP daemon as well as to preempt data flow from the HTTP daemon to the data transport module. In addition,

5   data transport information 814 may specify HTTP request data or HTTP response data that is to be transmitted to the client in accordance with the HTTP_OP field 806. More particularly, the data transport information 814 may include HTTP request/response DATA 816 (e.g., byte stream), a DIRECT field 818 that identifies data (e.g., file, shared memory segment,

10   previously cached object) that is directly accessible by the data transport module. Thus, this data need not be transported by the object from the HTTP daemon to the data transport module. In addition, the DIRECT TYPE 820 (e.g., file, shared memory segment, previously cached object) of the directly accessible data 818 may be specified in order to enable the data to be

15   subsequently accessed by the data transport module. Finally, a TRAILER 822 field may be used to include encapsulation information such as whether the response needs to be encapsulated prior to being transmitted to the client or indicate a method of encapsulation.

Caching attributes 824 (i.e., cache control indicators) may be provided

20   in the HTTP request-response object 802 by the HTTP daemon in order to manage information that is stored in the HTTP cache as well as to control transmission of the response data. As shown, the set of exemplary caching attributes 824 includes an advisory state 826, a nocache state 828, a CTAG 830, and an advise state 832. The advisory state 826 indicates whether the

25   cache manager 305 must communicate with the HTTP daemon 310 in order to determine whether response data can be transmitted to a client that has sent a HTTP request. In addition, the nocache state 828 indicates whether the HTTP

response and associated data are to be stored in the in-kernel HTTP cache 307. The CTAG 830 is a unique identifier associated with a HTTP response that enables the response to be associated with multiple HTTP requests in the HTTP cache. The advise state 832 may be provided by the HTTP daemon

5   310 in response to a HTTP request from the cache manager 305 as well as independently without receiving a request from the cache manager 305. The advise state 816 indicates an action to be taken with the response data and may specify a variety of actions, including but not limited to, modifying, storing, or flushing data from the in-kernel HTTP cache as well as controlling the

10   response data that is transmitted to a client that has submitted a HTTP request. Moreover, although the advise state 832 and the advisory state 826 are shown as separate states, they may be implemented as a single field. In addition, the HTTP daemon 310 may optionally provide response data in data field 816 in the HTTP request-response object.

15   In accordance with one embodiment, the NCA and the HTTP daemon exchange information through sending a HTTP request-response object in which the information is provided. Although the data transport module and HTTP daemon transmit the same type of object (e.g., HTTP request-response object), the data transport module and the HTTP daemon may transmit the

20   information in a variety of formats. Accordingly, the HTTP request-response object is merely illustrative and other mechanisms for storing and transmitting data between the data transport module and the HTTP daemon are contemplated.

Through the use of the present invention, the transmission of data that is provided in a HTTP request is accelerated. This is accomplished, in part,

25   is provided in a HTTP request is accelerated. This is accomplished, in part, through enabling efficient use of resources such as the CPU (e.g., threads) through providing HTTP request data for multiple HTTP requests in a single

stream. As described above, the HTTP request data for each HTTP request is identified through a unique connection identifier.

The present invention may be implemented on any suitable computer system. FIG. 9 illustrates a typical, general-purpose computer system 1502 suitable for implementing the present invention. The computer system may take any suitable form.

Computer system 1502 or, more specifically, CPUs 1504, may be arranged to support a virtual machine, as will be appreciated by those skilled in the art. The computer system 1502 includes any number of processors 1504 (also referred to as central processing units, or CPUs) that may be coupled to memory devices including primary storage device 1506 (typically a read only memory, or ROM) and primary storage device 1508 (typically a random access memory, or RAM). As is well known in the art, ROM acts to transfer data and instructions uni-directionally to the CPUs 1504, while RAM is used typically to transfer data and instructions in a bi-directional manner. Both the primary storage devices 1506, 1508 may include any suitable computer-readable media. The CPUs 1504 may generally include any number of processors.

A secondary storage medium 1510, which is typically a mass memory device, may also be coupled bi-directionally to CPUs 1504 and provides additional data storage capacity. The mass memory device 1510 is a computer-readable medium that may be used to store programs including computer code, data, and the like. Typically, the mass memory device 1510 is a storage medium such as a hard disk which is generally slower than primary storage devices 1506, 1508.

The CPUs 1504 may also be coupled to one or more input/output devices 1512 that may include, but are not limited to, devices such as video

monitors, track balls, mice, keyboards, microphones, touch-sensitive displays, transducer card readers, magnetic or paper tape readers, tablets, styluses, voice or handwriting recognizers, or other well-known input devices such as, of course, other computers. Finally, the CPUs 1504 optionally may be coupled

5   to a computer or telecommunications network, *e.g.*, an internet network or an intranet network, using a network connection as shown generally at 1514. With such a network connection, it is contemplated that the CPUs 1504 might receive information from the network, or might output information to the network in the course of performing the above-described method steps. Such

10   information, which is often represented as a sequence of instructions to be executed using the CPUs 1504, may be received from and output to the network, for example, in the form of a computer data signal embodied in a carrier wave.

In accordance with the present invention, the web server and

15   associated software may run on the CPUs 1504. Similarly, web pages and associated information may be stored in data storage devices 1506, 1508, and 1510.

Although illustrative embodiments and applications of this invention are shown and described herein, many variations and modifications are

20   possible which remain within the concept, scope, and spirit of the invention, and these variations would become clear to those of ordinary skill in the art after perusal of this application. For instance, the present invention is described as being implemented in a web server. However, the present invention may be used in other contexts. Moreover, the above described

25   process blocks are illustrative only. Therefore, the implementation of the cache manager, the data transport module, the NCA file system and HTTP daemon may be performed using alternate process blocks as well as alternate

data structures. Accordingly, the present embodiments are to be considered as illustrative and not restrictive, and the invention is not to be limited to the details given herein, but may be modified within the scope and equivalents of the appended claims.